Volume MLS Ray Casting

Christian Ledergerber, Gaël Guennebaud, Miriah Meyer, Moritz Bächer and Hanspeter Pfister, Senior Member, IEEE

Abstract—The method of Moving Least Squares (MLS) is a popular framework for reconstructing continuous functions from scattered data due to its rich mathematical properties and well-understood theoretical foundations. This paper applies MLS to volume rendering, providing a unified mathematical framework for ray casting of scalar data stored over regular as well as irregular grids. We use the MLS reconstruction to render smooth isosurfaces and to compute accurate derivatives for high-quality shading effects. We also present a novel, adaptive preintegration scheme to improve the efficiency of the ray casting algorithm by reducing the overall number of function evaluations, and an efficient implementation of our framework exploiting modern graphics hardware. The resulting system enables high-quality volume integration and shaded isosurface rendering for regular and irregular volume data.

Index Terms—Volume Visualization, Unstructured Grids, Moving Least Squares Reconstruction, Adaptive Integration

1 INTRODUCTION

Volume visualization has become an integral component of the scientific discovery process and the simulation pipeline. Data coming from scanning devices such as MRI or ultrasound machines, as well as results from simulations in fields like computational fluid dynamics, geophysics, and biomedical computing, are often represented volumetrically. These volumes, however, can be defined over regular or irregular grids, giving rise to several distinct classes of rendering algorithms and a plethora of implementations.

In this paper we propose a unified framework for generating high-quality visualizations of arbitrary volumetric datasets using the method of Moving Least Squares (MLS). MLS is a popular scheme for reconstructing continuous functions from scattered data due to its well understood mathematical foundations. Furthermore, the MLS framework contains several distinct components that provide a high level of control over the function reconstruction, and allows for a tunable amount of data smoothing as well as interpolation.

Using MLS, we reconstruct continuous functions from data stored on regular as well as irregular grids. We introduce volume MLS ray casting to generate high-quality images with volume integration and isosurfaces, such as that shown in Figure 1.

The volume MLS framework enables computing analytic derivatives for high-quality isosurface shading with specularities. To decrease the over-all number of MLS function evaluations we propose a novel adaptive preintegration technique.

The main contributions of this work are: a unified MLS framework for reconstructing continuous functions and their derivatives from both regular grids and unstructured volume data; a novel adaptive preintegration scheme for ray casting continuous functions with transfer functions that include isosurfaces; and a system that combines high-quality isosurface rendering and volume integration. We provide details on the theoretical foundations of the MLS framework, along with an exploration of the various tunable parameters. We also present an implementation of volume MLS ray casting on modern graphics processor units (GPUs), provide details of our system implementation, and show results on several regular and irregular volume data sets.

2 PREVIOUS WORK

Direct Volume Rendering: The generation of high-quality images of volume data relies on methods that accurately solve the volume rendering integral [28], which models a volume as a medium that can emit, transmit, and absorb light. The standard method for solving this integral is ray casting, and was proposed by Levoy [21] for regular grids, and by Garrity [14] for irregular grids, specifically tetrahedral meshes. Splatting, an alternative to ray casting, was proposed by Westover [49] for regular grids, and has been extended to irregular grids [25, 51]. In either case, a continuous function must be reconstructed from the discrete volume data.

While there are well-established mathematical frameworks for reconstruction of continuous functions from regular volume data [10, 18, 27, 9], irregular volumes are more challenging as there is no implied spatial structure to the data points. Ray casting of tetrahedral meshes allows a direct interpolation of the data at ray-triangle intersections using homogeneous coordinates [14]. For curvilinear grids, interpolation is more complicated [45] and is typically performed at the intersection points with cell faces [6] or in computational space [13]. Another popular algorithm for rendering tetrahedral meshes is the cell projection method [42], in which tetrahedral cells are projected in back to front order onto the image plane [5, 11]. However, a correct depth-ordering of cells does not always exist, and great care must be taken to avoid visual artifacts [50]. Finally, another class of algorithms uses a sweep-plane through the volume followed by rendering of 2D cells [15, 43].

In the case where no explicit structure of the data is given, scattered data reconstruction schemes are applied. Arguably the most prominent method among these are radial basis functions (RBFs) that were introduced to volume rendering by Jang et al.[17]. While RBFs can be evaluated and rendered efficiently [31], the method requires a costly initial solution to a global system of equations, the size of which is tied to the number of data points. Other methods proposed in the context of volume visualization are inverse distance weighting [35], which is a special case of MLS [41], and discrete Sibson interpolation [36]. This latter method inherently relies on resampling the data onto a regular grid. In contrast to these approaches, volume MLS ray casting provides a unifying mathematical framework for the reconstruction of arbitrarily sampled volume data that allows the user to control the reconstruction and data smoothing in a principled manner.

MLS Function Reconstruction: MLS was introduced almost half a century ago by Shepard [41], and more recently shown by Levin as a powerful surface approximation technique [20]. This technique has been embraced by the computer graphics community for reconstructing point set surfaces [3, 4], where a number of extensions have been proposed, such as reconstructing surfaces with sharp features [12, 23] and using anisotropic support regions [1]. Unlike RBFs, MLS does not require a solution to a global system of equations.
Volume Integration: During ray casting, a transfer function (TF), which maps scalar values and (optionally) gradients to colors and opacities, is applied to the volume function. These color and opacity properties of the volume are integrated along rays to obtain the final pixel colors. The TF must be smooth in order to compute the numerical integral efficiently and accurately as no general, closed-form solutions are available. For the case of a smooth TF, well-known adaptive numerical schemes can be employed [32], and theory from Monte Carlo integration can guide their design [7]. Also, interval arithmetic has been proposed to make the above approaches robust [33].

In practice, however, smooth TFs limit the range of achievable rendering effects, e.g., isosurface rendering via thin spikes in the TF that, in theory, require an infinite sampling density. To circumvent this problem Roettger et al. [39] introduce preintegration, which avoids visual artifacts by precomputing the volume rendering integral for ray segments, assuming that the underlying scalar function is linear. Recent efforts focus on accelerating the precomputation [8, 37], faster approximation algorithms [24], multidimensional transfer functions [30], and preintegrated lighting [24, 29].

In this paper we propose an adaptive integration scheme based on preintegration. This scheme is similar in spirit to other numerical integration schemes, such as adaptive Simpson [32] and the second derivative refinement scheme of Roettger et al. [38].

3 Volume MLS

In order to generate high-quality images, a ray caster must evaluate a continuous representation, and the associated derivatives, of volumetric data. For volume data defined over a grid, two approaches can be taken: first, the connectivity of the grid can be used to define a reconstruction scheme over the locally connected data points; or second, the data points can be used independently of their grid connections in a scattered data reconstruction scheme. The first approach works well for regular grids, where higher order polynomials are often employed to reconstruct functions of arbitrary continuity. For irregular grids, however, reconstructions are challenging for anything other than regularly shaped mesh elements. This flexibility provides the capacity to handle poorly shaped mesh elements. This flexibility provides the capacity for (controllably) handling noise due to measurement errors in scanned data or numerical errors in simulated data. Furthermore, the MLS approximation is a continuous reconstruction with well-defined, smooth derivatives which allows for high-quality shading effects.

In this section, we first describe the MLS framework and mathematical foundations in the context of reconstructing continuous functions from scattered data. We then discuss and explore the various components of the MLS method as they relate to reconstructing data stored over regular and irregular grids specifically for generating high-quality visualizations by ray casting. Our notation is as follows: bold face lower-case variables denote column vectors, such as \( \mathbf{x} = [x \ y \ z]^T \), while bold face upper-case variables denote matrices, such as the identity matrix \( \mathbf{I} \). We note that the MLS discussion and mathematical definition holds for arbitrary dimensions, even though we restrict our examples and results to \( \mathbb{R}^3 \).

3.1 The MLS Framework

The main idea behind the MLS reconstruction method is depicted for a 1D signal in Figure 2. Shown in red, a continuous approximation \( f(x) \) is reconstructed from a set of \( n \) data points \( \{(x_i, f_i)\} \) by computing

\[
f(x) = g_x(x),
\]

and evaluating a local approximation \( g_x \) at \( x \) (shown in blue):

\[
f(x) = g_x(x).
\]

The local approximation \( g_x \) is obtained by the following WLS minimization problem:

\[
g_x = \arg \min_g \sum_i w_i(x_i) |g(x_i) - f_i|^2,
\]

where \( w_i(x_i) \) is the weight of the sample \( p_i \) for the current evaluation point \( x \). These weights typically decrease with distance from \( x \) (see Section 3.1.2). Assuming \( g_x \) can be represented as a linear combination of \( k \) basis functions \( \mathbf{b}(x) = [b_1(x), \ldots, b_k(x)]^T \) with coefficients \( \mathbf{c}_x = [c_1, \ldots, c_k]^T \), Equation 2 can be rewritten in the following matrix form:

\[
\mathbf{c}_x = \min_{\mathbf{c} \in \mathbb{R}^k} \| \sqrt{\mathbf{W}(x)} \mathbf{B} \mathbf{c} - \mathbf{f} \|^2,
\]

where \( \mathbf{W}(x) = \text{diag}(w_1(x), \ldots, w_n(x)) \in \mathbb{R}^{n \times n} \) and \( \mathbf{B}_{ij} = b_i(x_j), \mathbf{B} \in \mathbb{R}^{n \times k} \). It is well-known from linear algebra that the above least squares problem can be solved using the normal equation. Substituting the solution for \( \mathbf{c}_x \) into \( f(x) = \mathbf{b}(x)^T \mathbf{c}_x \) yields the following explicit formulation of the MLS approximating function:

\[
f(x) = \mathbf{b}(x)^T \left( \mathbf{B}^T \mathbf{W}(x) \mathbf{B} \right)^{-1} \mathbf{B}^T \mathbf{W}(x) \mathbf{f}.
\]

This function is smooth, assuming the weights \( w_i(x) \) are smooth [19], and can be computed as long as the weighted covariance matrix \( \mathbf{B}^T \mathbf{W}(x) \mathbf{B} \) is invertible, which is the case if \( \text{rank}(\mathbf{W}(x) \mathbf{B}) \geq k \) — this
implies that at least $k$ weights have to be nonzero. We emphasize that the matrix $W(x)$ depends on the evaluation point, and hence the least squares system must be solved at every $x$.

In the MLS framework, the smoothness of the reconstruction, as well as the tightness of the approximation to the data values (i.e., interpolation), is controlled through the choice of the basis and weighting functions. Each of these components are discussed in the following sections.

### 3.1.1 Basis Functions

When choosing the basis (i.e., $b(x)$), the main trade-offs are approximation power versus efficiency. Lower order polynomial bases are less accurate at approximating the data due to fewer degrees of freedom in the function. Higher order polynomial bases, however, are less efficient as a larger linear system must be solved, as well as the determination of a larger neighborhood to maintain stability in the linear system computation. Furthermore, a too-large basis can yield over-fitting. We have experimented with several bases, and for ray casting MLS volumes, we advocate the use of linear $b(x) = (1, x, y, z)^T$ basis functions. Indeed, as shown in Figure 3, a quadratic basis does not give much better results while increasing the computational burden by more than a factor of two.

### 3.1.2 Weighting Functions

Once the basis function has been fixed, the quality of the reconstruction highly depends on the weights assigned to the neighboring samples. For uniform data sets, we employ the following simple weighting scheme:

$$w_i(x) = \theta \left( \frac{\|x - x_i\|}{h} \right),$$  \hspace{1cm} (5)

where $h$ is the support size, and $\theta$ is a smooth, decreasing weight function for which several options have been proposed in the literature [47]. In this work, we use the following compactly supported polynomial:

$$\theta(x) = \begin{cases} 
(1 - x^2)^\alpha & \text{if } x < 1 \\
0 & \text{otherwise.} 
\end{cases}$$  \hspace{1cm} (6)

where the power factor $\alpha$ controls the speed at which the weighting function decreases. Although larger values of $\alpha$ increases the continuity of the function, large values also require a wider support for stable computation. A good compromise is $\alpha = 4$, which guarantees a $C^0$ reconstruction. Note that in Equation 4, since $\theta(x)$ has a finite support, only points within a distance $h$ of $x$ have to be taken into account, which allows for efficient computation while avoiding square roots in the distance computations. The effect of the support size in the reconstruction is depicted in Figure 3.

To reconstruct irregular, and possibly anisotropic, datasets we use the weighting scheme proposed by Adamson and Alexa [1], and attach ellipsoidal weight supports $H_i$ to each input sample:

$$w_i(x) = \theta \left( \|H_i(x - x_i)\| \right).$$  \hspace{1cm} (7)

The weight transforms, $H_i$, are obtained using a covariance analysis of the one-ring neighborhoods. Figure 6 depicts an example of such anisotropic weight supports. For irregular datasets that do not present significant anisotropy, it is sufficient to replace the expensive ellipsoids by balls, reducing the $3 \times 3$ matrices $H_i$ to a single scalar radius $h_i$.

In addition to data approximation, the MLS framework is also capable of data interpolation using weighting functions that contain a singularity [41]. One such function proposed by Adamson et al. [2] is:

$$\theta(x) = \begin{cases} 
\ln(x)^2 - x^2 + 2x - 1 & 0 \leq x \leq 1 \\
0 & x > 1 
\end{cases}$$  \hspace{1cm} (8)

Compared with other interpolating weighting schemes, such as those based on inverse distances [41, 35], Equation 8 has a finite support, which is important for efficient evaluation. One property of interpolating MLS which is noteworthy is the flat spot phenomenon [41, 19]. This implies that for a constant basis, the gradient of the approximation to any data point is zero $\nabla f(x_i) = 0$. To avoid this undesirable characteristic a higher order basis can be used.

### 3.2 Gradient Computation

The generation of high-quality shading effects requires accurate gradient information, as does the adaptive preintegration scheme discussed in Section 4. There are two established methods for computing the gradient within the MLS framework. The most accurate method is to analytically compute the gradient from the partial derivatives of the MLS scalar field formula (Equation 4):

$$\frac{\partial f(x)}{\partial x_k} = \frac{\partial b(x)^T}{\partial x_k} \mathbf{c}_k - \mathbf{b}^T \mathbf{A}(x)^{-1} \left( \frac{\partial \mathbf{A}(x)}{\partial x_k} \mathbf{c}_k + \mathbf{B}^T \frac{\partial W(x)}{\partial x_k} \mathbf{f} \right)$$  \hspace{1cm} (9)

where $\mathbf{A}(x)$ is the weighted covariance matrix $\mathbf{B}^T \mathbf{W}(x) \mathbf{B}$. 

---

Fig. 3. The effects of using different basis functions and support sizes for reconstructing the Marschner-Lobb data set. In the first row, the data is approximated using a linear basis with support size 1.8 (left), 2.4 (middle), 4 (right). The second row shows a quadratic basis with support size 2.6 (left), 3 (middle), 4 (right). The boundary behavior of the MLS reconstruction is due to the divergence of the function during extrapolation [19].
In the remainder of this section we first formulate an approximation of the volume rendering integral using arbitrary length evaluation intervals (Section 4.1), which is followed by a description of the interval length adaptation scheme (Sections 4.2 and 4.3). We then include details on the computation of the preintegration tables and a description of how to handle clipped ray segments incurred from intersecting rays with irregular grids (Section 5).

4.1 Approximating the Volume Integral

In this section we derive an approximation of the volume rendering equation [28] for use with the proposed adaptive preintegration method. Specifically, the derivation allows for the integration along a ray to be approximated with a composition of colors and opacities computed over arbitrary-length subintervals of the ray. To begin, the volume rendering integral is given as:

\[ I = \int_0^D c(f(\lambda)) \tau(f(\lambda)) e^{-\int_0^\lambda \tau(f(\lambda'))} d\lambda' d\lambda \]  

where \( c \) is the emitted color and \( \tau \) is the extinction which changes with the scalar function \( f \) along the ray \( x(\lambda) \). For use with preintegration, the domain \( D \) of the integral can be split into the intervals \([d_0, d_1, \ldots, d_{n-1}, d_n]\), where \( d_0 = 0 \) and \( d_n = D \). the opacity \( (\alpha) \) and color \( (C) \) for arbitrary length intervals can then be defined as:

\[ \alpha_{[d_i, d_{i+1}]} = 1 - \exp \left( - \int_{d_i}^{d_{i+1}} \tau(f(x(\lambda))) d\lambda \right) \]  

(12)

\[ C_{[d_i, d_{i+1}]} = \int_{d_i}^{d_{i+1}} c(f(x(\lambda))) \tau(f(x(\lambda))) e^{\int_{d_i}^\lambda \tau(f(x(\lambda'))) d\lambda'} d\lambda \]  

(13)

Using Equations 12 and 13 we arrive at the well-known recursive front-to-back composition formulas for opacity and color:

\[ \alpha_{i+1} = \alpha_i + (1 - \alpha_i) \alpha_{[d_i, d_{i+1}]} \]  

(14)

\[ C_{i+1} = C_i + (1 - \alpha_i) C_{[d_i, d_{i+1}]} \]  

(15)

where \( I = C_n + \lambda_0 \alpha_0 \) and \( \alpha_0 = 0.0 \). This reformulation is exact, and because there are no assumptions about the length of the intervals, the step size can be varied along the ray. Furthermore, as Equations 14 and 15 add the contribution of the interval \([d_i, d_{i+1}]\), this contribution is fully determined by \((1 - \alpha_i) C_{[d_i, d_{i+1}]}\) and \((1 - \alpha_i) C_{[d_i, d_{i+1}]}\). This property is important in Section 4.2.

Assuming that \( f(x(\lambda)) \) is a linear function on the interval \([d_i, d_{i+1}]\), \( \alpha_{[d_i, d_{i+1}]} \) and \( C_{[d_i, d_{i+1}]} \) can be precomputed in a 3D table. This precomputation depends on the value of the scalar function \( f \) at the front \( f(d_i) \) and the back \( f(d_{i+1}) \) of the interval, as well as on the length \( l = d_{i+1} - d_i \) of the interval. We note that these table values are only an approximation to the true integral over the interval because the function is, in general, nonlinear and the table has a finite resolution.

For the remainder of this section let us denote the approximated color of an interval (stored in a preintegration table) by \( \tilde{C}_{i, i+1} \) and similarly, the approximated opacity as \( \tilde{\alpha}_{i, i+1} \).

4.2 Interval Length Adaptation

The accuracy of the volume integral approximation along a interval can be increased by further subdivision along the interval. As such, the interval can be progressively refined until the difference of the volume integral approximation between any two successive subdivision levels is below some error threshold.

Starting with a coarse step size, the initial volume integral approximation along an interval for color (i.e., the contribution of the interval in Equation 14), \( C_1 \), is:

\[ C_1 = \tilde{C}_{i, i+1} \]  

(16)
To increase the accuracy of this approximation, the interval is subdivided, and the contribution, $C_2$, becomes a summation over the two subintervals:

$$C_2 = \frac{1}{2} \left( (1 - \alpha_{i+1/2}) C_{i+1/2} + (1 - \alpha_i) C_i \right)$$

The subdivided contributions for opacity are defined similarly as:

$$\alpha_{i+1/2} \approx \left( \alpha_{i+1/2} + (1 - \alpha_{i+1/2}) \alpha_{i+1/2} \right)$$

This subdivision continues recursively until the difference in the volume integral approximation over the entire interval changes by only a small amount.

Based on the volume integral approximations $C_1$ and $C_2$, we define the relative error of the approximation as $\frac{|C_1 - C_2|}{|C_1|}$, where $C_1 = \sqrt{C_1^2 + C_2^2}$ — note that this is not independent from the opacity (see Equations 15 and 16). To adapt the sampling as the opacity increases we multiply the relative error by $(1 - \alpha)$. The final termination criteria for the recursive subdivision is:

$$\left( 1 - \alpha \right) |C_1 - C_2| / |C_1| < \epsilon.$$  (17)

In the recursion it is important to first subdivide the interval $[d_i, d_{i+1}]$ such that $\alpha_{i+1/2}$ is known when subdividing the interval $[d_{i+1/2}, d_{i+1}]$. This is because $\alpha_{i+1/2}$ is used in the computation of the termination criteria for the interval $[d_{i+1/2}, d_{i+1}]$. Note that this scheme is automatically optimal for constant transfer functions and linear scalar fields, since $C_1$ and $C_2$ will be equal to within numerical errors.

### 4.3 Handling Scalar Function Extrema

The subdivision scheme in the previous section works well as long as the scalar function is monotonic over an interval. If this property is violated, an isosurface may exist near an extrema of the scalar function. To be able to explicitly control the accuracy of the preintegration tables we improved the incremental subrange preintegration such that we can control the maximal step size of the numerical integration, i.e., we supersample intervals if necessary. While the table of length $l_c$ can also be computed from the table of length $l_i/2$ using the idea of incremental preintegration [46], we found it difficult to control the errors in such a scheme.

The need for high accuracy preintegration also implies larger tables. All the results in this paper use tables with a resolution of 512^2. While these large tables increase the memory footprint of the system (the tables require approximately 72MB), they are relatively fast to compute (up to 10 seconds on the CPU) and could likely be generated on the GPU at faster speeds, enabling interactive user control of the TFs.

In the case where an interval is clipped against the bounding geometry of the volume we must compute the integral for a length that is not present in any of the tables. For this scenario, the integral can be determined by carefully combining values from different tables. Because this happens only twice per ray, this additional complexity is virtually unnoticeable.

Next, the ray caster intersects each ray with the boundary of the volume, finding all the intervals that lie within the (possibly nonconvex) hull of the grid. These intersection points are important as the MLS function approximation is ill-defined away from the data points. Among all of the intersected ray segments, integration intervals are determined using the adaptive scheme described in Section 4, and the MLS function and gradient are computed at the front and back of each interval (see next section). Finally, the volume integration along the interval is computed via lookups into the preintegration tables. The resulting color and opacity are composited with the stored values on the ray. Early ray termination occurs when the opacity value is greater than some threshold close to one (for the results in this paper we use 0.99).

### 5.1 Fast and Robust MLS Evaluation

The most time critical part of our rendering system is the evaluation of the MLS function and its gradient, which involves both a neighborhood query and solving a linear system. The complexity of the latter operation depends on the size of the neighborhood and the number of degrees of freedom in the basis. For a small basis, once the covariance matrix $B = \mathbf{W} \mathbf{B}^T$ is computed, solving the normal equation itself is relatively cheap and can be done efficiently using a Cholesky decomposition without square roots of the covariance matrix. Given

$$B = LDL^T.$$  (20)

where $L$ is a lower triangular matrix with unit diagonal, and $D$ is a diagonal matrix, we can solve $c_x$ as:

$$c_x = L^{-T} D^{-1} L^{-1} \left( B^T \mathbf{W} \mathbf{f} \right).$$  (21)

Not only does avoiding the computation of square roots improve performance, but it also increases the stability of the system. Moreover, both stability and efficiency can be further improved by shifting the origin of the basis to the the evaluation point, i.e., $b(x) = b(x-x_0)$, which is mathematically equivalent and avoids large numbers when computing the function far from the origin. Since the basis function is only evaluated at $x$, i.e., at the origin, it is therefore sufficient to compute the constant coefficient $c_1$ of $c_x$, hence avoiding most of the calculation in Equation 21.

The computations of the neighbor samples, on the other hand, requires efficient spatial-data structures. More precisely, given an evaluation point $x$, the challenge is to find all samples $p_i$ having strictly positive weights. Therefore, the choice of both the data structure and search algorithm is related to the weighting scheme. For the weighting function defined in Section 3.1.2, we need to find all the ellipsoids (or balls) containing $x$. In our CPU implementation this is accomplished using a kd-tree partitioning of the space where a sample is referenced by all leaves intersecting its corresponding ellipsoid. Our kd-tree is built recursively from top to bottom until the size of the node is smaller than the average radius of the node’s samples.
In this section we present results of the proposed MLS framework for
regular grids as shown in the renderings of the heptane plume dataset in
Figure 1 and the Marschner-Lobb dataset in Figure 7. To illustrate MLS ray
casting of curvilinear grids we show renderings of the highly anisotropic NASA
blunt fin dataset in Figure 6. We show performance numbers for render-
ing the blunt fin on the CPU because we did not yet implement anisotropic
weights on the GPU. Note that the CPU implementation is not optimized and
takes orders of magnitude longer than the GPU implementation. Irregular grid renderings of tetrahedral meshes are
shown in Figure 8 with the buoyy ball dataset (left) and the Utah torso
model (right). For the torso model we estimated the sampling density
at the input samples using the $k$-nearest neighbor radius and rendered
the reconstructed function. In all of these images, the high-quality
shading of smooth isosurfaces confirms the quality of the MLS recon-
struction, while the range of datasets and grid-types illustrates the
versatility of the MLS scheme.

5.2 CUDA Implementation

In order to achieve nearly interactive performance we implemented a
GPU-based version of the raycaster using CUDA [34]. In our cur-
current implementation, each thread processes a single ray, and the ray
traversal is entirely decoupled from the volume data structure. For the
ray traversal we implemented both a constant step size strategy and
the adaptive algorithm described in Section 4. In the latter case, the
recursion is managed via a small stack stored in local memory. To ef-
ciently find neighbor samples, we use the dynamic redundant octree
of Guennebaud et al. [16]. This spatial search data structure is partic-
ularly efficient for GPU applications, allowing for dynamic updates as
well as intuitive control of the memory footprint versus performance.
We did not yet implement the anisotropic weights discussed in Sec-
tion 3.1.2 on the GPU, even though that is possible.

6 Results

In this section we present results of the proposed MLS framework for
several well-known regular and irregular datasets. Except where noted
otherwise, all of the MLS images were generated using a linear basis
with approximate gradients and the proposed adaptive preintegration
scheme with our GPU-based implementation.

Table 1 provides information on each data set, including the rele-
vant parameters used to generate the images and performance results.
The results were computed on a 2.6GHz Intel Core 2 Duo with 8 GB
of RAM and an NVIDIA Quadro FX5600 GPU with 1.5 GB of video
memory. The parameters in Table 1 lead to a root mean square error
(RMSE) below 5% compared to ground truth images computed us-
ing a very small step size. We found that this error tolerance leads
to visually indistinguishable results from the ground truth images. Of
all the parameters $\epsilon$ (see Section 4.2) is the least sensitive for the
resulting image quality, while a too-large value for $l_{\text{max}}$ (see Section 5)
can cause the adaptive integration scheme to entirely miss extrema in
the volume integral by inducing an early termination of the automatic
refinement scheme. This early termination results in visible discontinu-
ities in the final image. A too-large segment length for the standard
integration scheme also introduces artifacts, but those are harder to
detect because the images is less likely to exhibit discontinuities as the
sampling positions on neighboring rays are very similar.

The capabilities of the MLS system for regular grids are shown in
the renderings of the heptane plume dataset in Figure 1 and the
Marschner-Lobb dataset in Figure 7. To illustrate MLS ray casting of
curvilinear grids we show renderings of the highly anisotropic NASA
blunt fin dataset in Figure 6. We show performance numbers for ren-
dering the blunt fin on the CPU because we did not yet implement anisotropic
weights on the GPU. Note that the CPU implementation is not optimized and takes orders of magnitude longer than the GPU
implementation. Irregular grid renderings of tetrahedral meshes are
shown in Figure 8 with the buoyy ball dataset (left) and the Utah torso
model (right). For the torso model we estimated the sampling density
at the input samples using the $k$-nearest neighbor radius and rendered
the reconstructed function. In all of these images, the high-quality
shading of smooth isosurfaces confirms the quality of the MLS recon-
struction, while the range of datasets and grid-types illustrates the
versatility of the MLS scheme.

To compare the MLS system with existing ray casting methods,
we implemented a standard piecewise-linear scheme with weighted
normals [26] for tetrahedral meshes (see Figure 6 (left)). The image
shows significant visual artifacts, even though we are using weighted
normals. Figure 7 (middle) shows the result of cubic B-spline convo-
lution for the reconstruction of a continuous function over a regular
grid. The B-spline reconstruction has been shown to be optimal for
the Marschner-Lobb dataset [27]. The comparison shows that MLS
reconstruction obtains competitive results. Using a linear MLS ba-
sis on a regular grid does not reduce approximation errors at the grid
points [44], which explains why the MLS approximation is not tighter.
However, we note that convolution is equivalent to MLS with a con-
stant basis function [44], making MLS is a more general framework
that subsumes convolution.

6.1 Performance Analysis

In theory, the complexity of our algorithm is $O(m \ast p \ast (k + \log(n)))$ for irregular data sets where $m$ is the number of pixels, $p$ is the number
of function evaluations per ray, $n$ is the size of the dataset, and $k$ is the
number of neighbors used. As pointed out in Section 4.2, the number
of function evaluations per ray depends on the complexity of the scalar
function and transfer function. The runtime of a function evaluation is
$O(k + \log(n))$, where $O(\log(n))$ represents the traversal complexity of
the hierarchical data structure — in practice the size of the dataset has
very little impact on the evaluation cost that is highly dominated by the
size of the neighborhoods. Using an adaptive scheme, the total num-
ber of function evaluations can be significantly reduced (see Table 1).
However, the limited flexibility of current GPU architectures makes it
very challenging to take full advantage of this gain. In our experiments
we observed that more complex datasets (i.e., those that contain more
high-frequency features) benefited the most from the adaptive scheme.

To give further insight into the complexity of a function evaluation
we compare the number of floating point operations (flops) used for

![Image](image-url)
the evaluation of a number of reconstructing functions in Table 2. We compare standard trilinear interpolation of the normals and standard tetrahedral ray casting with weighted normals [26] to MLS ray casting with constant and linear basis functions. The MLS performance is a function of the number of neighbors \(k\). In practice, \(k\) usually varies between 8 and 20. From this analysis we conclude that the fflods for one MLS evaluation is usually less then three times that of a trilin-ear interpolation. For irregular datasets the most expensive part of a function evaluation remains the neighborhood query.

Table 2. This table compares the number of flops needed to compute one function evaluation for different function approximation schemes and \(k\) neighbor samples. For the constant basis we computed the analytic MLS gradient while we included the approximate gradient for the linear basis. For the trilinear interpolation we included a central difference gradient.

<table>
<thead>
<tr>
<th></th>
<th>TriLinear Kernel</th>
<th>Standard Tetrahedral Ray Casting</th>
<th>Constant Basis</th>
<th>Linear Basis</th>
</tr>
</thead>
<tbody>
<tr>
<td>function</td>
<td>30</td>
<td>7</td>
<td>15(k + 1)</td>
<td>39(k + 56)</td>
</tr>
<tr>
<td>gradient</td>
<td>186</td>
<td>20</td>
<td>14(k + 9)</td>
<td>free</td>
</tr>
<tr>
<td>total</td>
<td>216 (+52 ray intersect.)</td>
<td>29(k + 10)</td>
<td>39(k + 56)</td>
<td></td>
</tr>
</tbody>
</table>

6.2 Reconstruction Quality Analysis

In practice, function reconstruction is an ill-posed problem for which there are many reasonable solutions. Furthermore, for visualization there is a tradeoff between approximation power and visually pleasing results. The latter is subjective, while the former is well-studied for MLS. Convergence theory [48, 22] yields error bounds of the MLS reconstruction, and multivarious nonparametric regression in statistics [40] has lead to a thorough understanding of its bias and variance.

7 Conclusions & Future Work

Volume MLS ray casting produces high-quality images and provides an easy to understand mathematical framework with intuitive parameters. It is a first step towards creating an efficient framework which is able to handle any kind of data regardless of its topology. A next step in this direction would be to remove the necessary boundary information and compute a (possibly) nonconvex hull of the data purely based on the sampling. Our volume MLS framework ties in nicely with a wealth of existing tools for MLS reconstruction. For example, one could extend this work to include exact reconstruction of sharp features.

The proposed system lends itself well to streaming data. Since an additional data point only affects the scalar function in a bounded region, a local update of the image could be computed and no global system has to be solved. The ray casting approach is attractive due to its simplicity and because it can easily be extended to include complex cut and region of interest geometry. Finally, our adaptive preintegration scheme is immediately useful for other ray casting frameworks. We believe that progress on the theory of the convergence of preintegration could improve the efficiency and robustness of our scheme.

Acknowledgements

The authors sincerely thank Markus Gross, ETH Zurich, for inspiring this work during stimulating discussions at the outset of this project. The authors wish to thank David Luebke and NVIDIA for generous donation of hardware. Christian Ledergerber, Miriah Meyer, and Moritz Bücher were supported by funding from the Initiative in Innovative Computing (IIC) at Harvard. Gael Guennebaud was supported by the ERCIM “Alain Bensoussan” Fellowship Programme. The Utah torso model is courtesy of the NCRR Center for Integrative Biomedical Computing at the University of Utah, and heptane plume dataset is courtesy of the Center for the Simulation of Accidental Fires and Explosions, also at the University of Utah.